

Overstapgids

Van Swift 1 naar Swift 2



Roelf Sluman



de iOS Academie

Versie 1.1 (oktober 2015)

© 2015 Roelf Sluman / De iOS Academie. Mits in ongewijzigde vorm, mag deze uitgave zonder voorafgaande schriftelijke toestemming van de auteur worden openbaar gemaakt of verveelvoudigd. Met andere woorden: verspreiden maar :)

Inhoud

Inleiding 4

1 Je programmacode omzetten met Xcode 6

2 Wijzigingen in Xcode 12

3 Wijzigingen in Swift 18

4 Je volgende stap 32

Inleiding

Heb je al eens in Swift geprogrammeerd? En heb je inmiddels de nieuwste versie van Xcode geïnstalleerd? Dan heb je ongetwijfeld al gemerkt dat er flink wat is veranderd. Xcode werkt anders, Swift is flink op de schop gegaan en op het eerste gezicht raak je flink in de war door alle wijzigingen.

Speciaal daarvoor is deze Overstapgids bedoeld. Hij helpt je om snel over te schakelen van Swift 1 en Swift 2.

Wil je de Playgrounds niet zelf intypen? Je vindt ze op deze pagina:

www.iosacademie.nl/overstapgids-swift-2

Enkele gedeelten en Playgrounds uit deze Overstapgids zijn afkomstig uit de populaire serie eBooks *Apps bouwen met Swift - programmeren met Swift*. Voor meer informatie daarover kun je terecht op deze pagina:

www.iosacademie.nl/apps-bouwen-met-swift

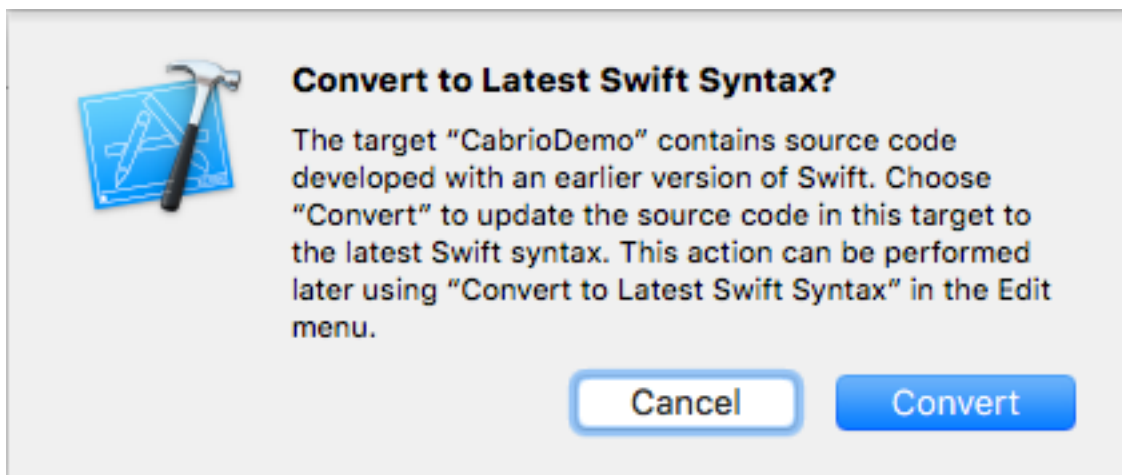


Veel succes met je overstap van Swift 1 naar Swift 2!

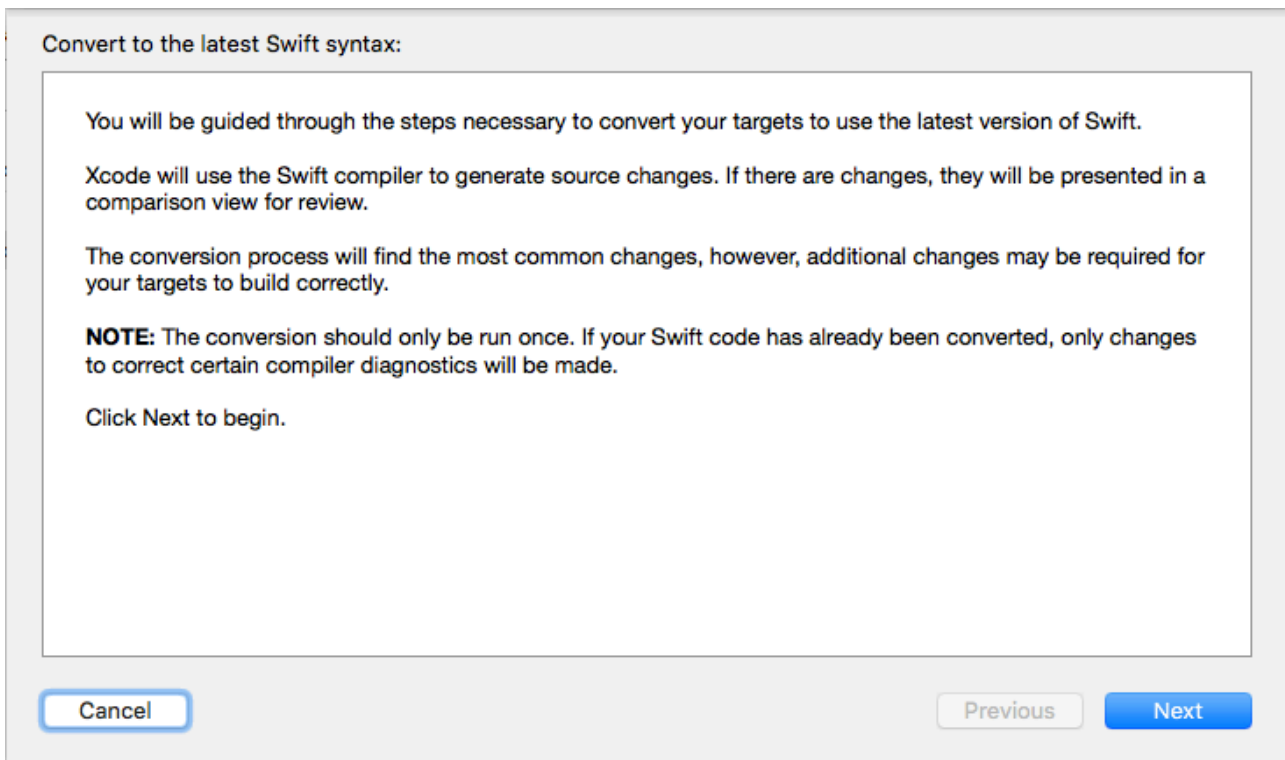
1 Je programmacode omzetten met Xcode

Heb je al eens in Swift geprogrammeerd? En heb je inmiddels de nieuwste versie van Xcode geïnstalleerd? Dan heb je ongetwijfeld al gemerkt dat er flink wat is veranderd. De *syntax* van Swift is gewijzigd: sommige functies en sleutelwoorden bestaan niet meer of zijn vervangen door andere, waardoor programmacode die in Swift 1 is geschreven, moet worden omgezet naar Swift 2.

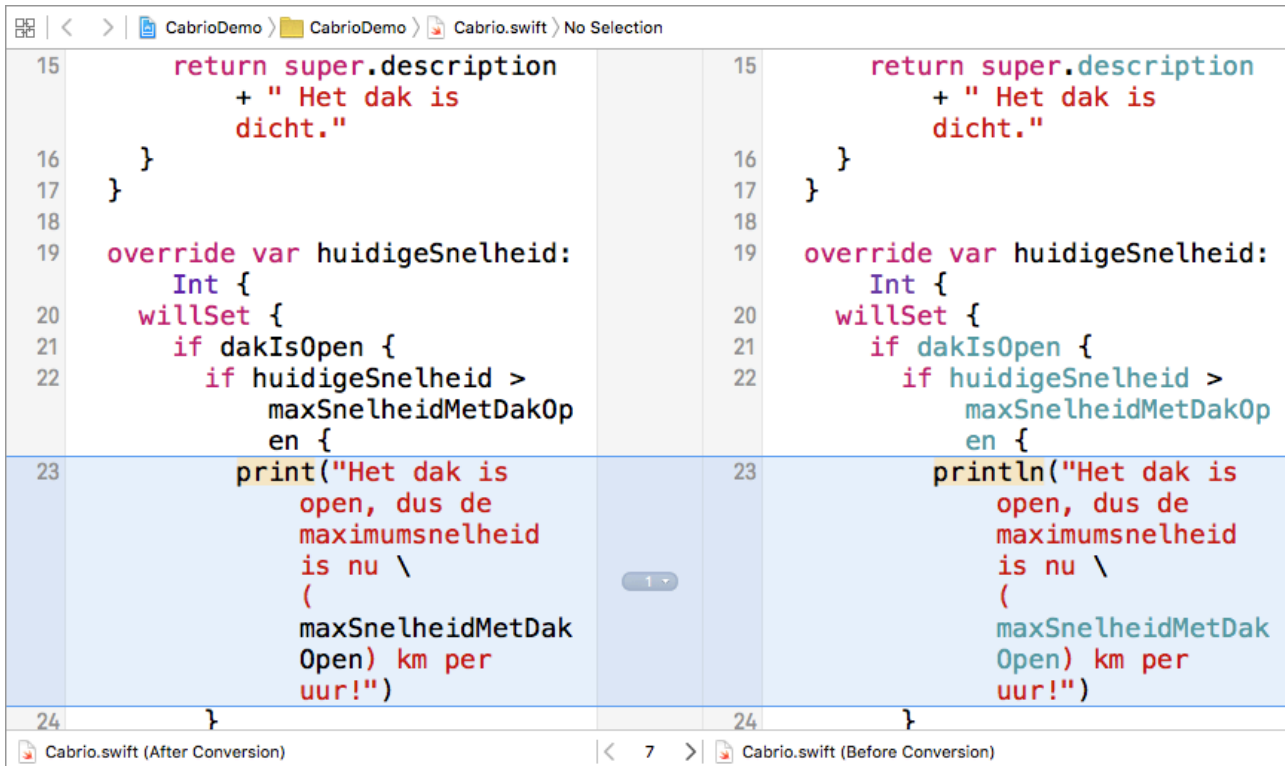
Gelukkig kan Xcode de meeste programmacode voor je omzetten. Zodra je een app probeert te laden die je met een oudere versie van Xcode hebt gemaakt, krijg je de volgende melding te zien:



Wanneer je op *Convert* klikt, vertelt Xcode je wat het gaat doen: het analyseert je app en laat je daarna alle programmacode zien die is verouderd en die moet worden omgezet naar Swift 2.



Wanneer je op *Next* hebt geklikt, gaat Xcode aan het werk. Als er verouderde code is aangetroffen, krijg je twee versies te zien: rechts zie je de oude programmacode en links de nieuwe. Klik op *Save*, waarna je project wordt bijgewerkt.

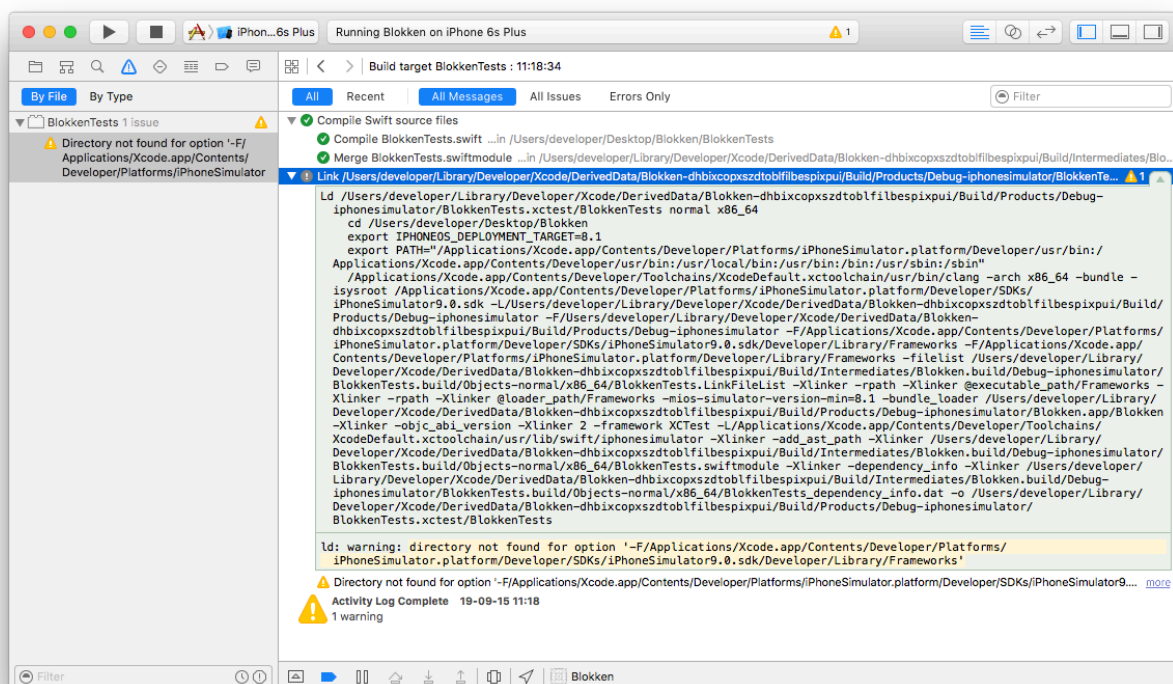


In de meeste gevallen ben je dan klaar. Xcode kan echter niet alle problemen oplossen die zijn ontstaan door wijzigingen in Swift. Ook kunnen Playgrounds door Xcode 7 niet automatisch worden omgezet. In zulke gevallen zul je dus zelf aan de slag moeten. Op de volgende pagina's lees je daar meer over.

De foutmelding: :(null): Directory not found for option

Nadat je een 'oude' app naar Xcode 7 hebt omgezet, kan het gebeuren dat je, zodra je probeert je app te starten, een storende waarschuwing ziet:

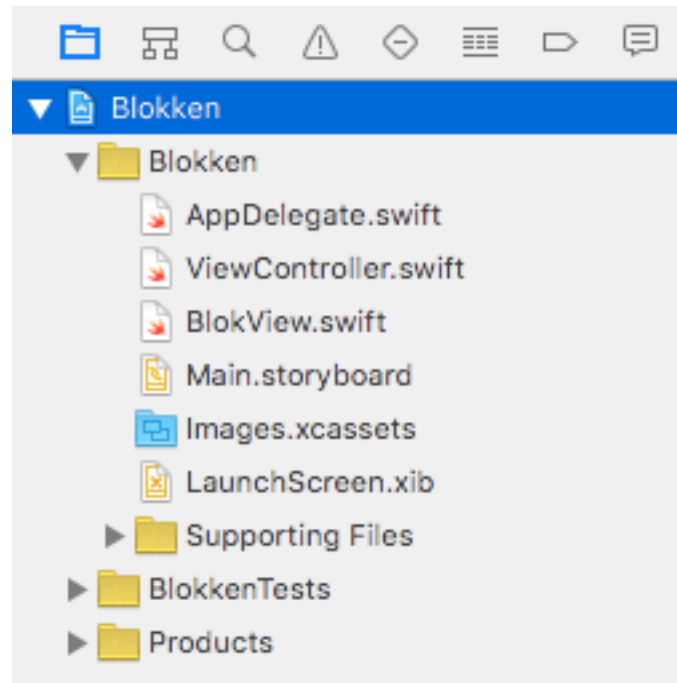
:(null): Directory not found for option '-F/Applications/Xcode.app/Contents/Developer/Platforms/iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator9.0.sdk/Developer/Library/Frameworks'



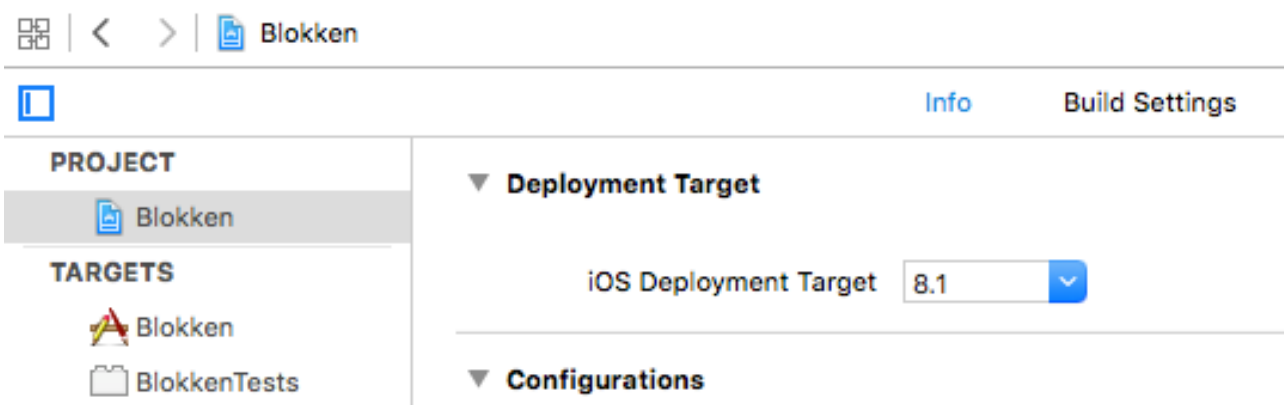
Deze waarschuwing verschijnt omdat Xcode per ongeluk een instelling overneemt in een (nieuwe) *target*, bedoeld om Xcode automatisch te laten testen.

Om deze waarschuwing te verwijderen, doe je het volgende:

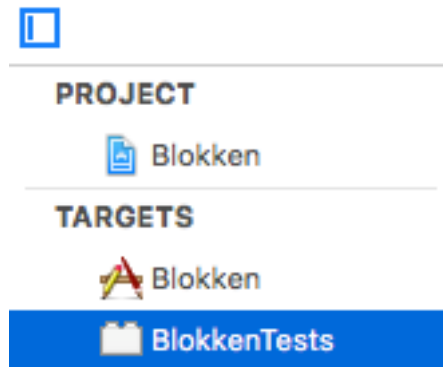
Klik, in de Navigator, op het project-icoon van je app:



In de editor verschijnt informatie over je project. Links daarvan staat de *Projects and targets list*:



Klik, in de *Projects and targets list*, op de target die eindigt op *Tests*. In de afbeelding hierna is dat *BlokkenTests*, maar in jouw project staat hier natuurlijk de naam van jouw app, gevolgd door *Tests*.



Klik, in de editor, op de *Build Settings*-tab:

| < | > | Blokken

 General Resource Tags Info **Build Settings** Build Phases Build Rules

PROJECT

Blokken

TARGETS

Blokken

BlokkenTests

Basic **All** Combined Levels +

▼ Architectures

Setting	BlokkenTests
Additional SDKs	
Architectures	Standard architectures (armv7, arm64) - \$
Base SDK	Latest iOS (iOS 9.0) ↕
▼ Build Active Architecture Only	<Multiple values> ↕
Debug	Yes ↕
Release	No ↕
Supported Platforms	iOS ↕
Valid Architectures	arm64 armv7 armv7s

▼ Assets

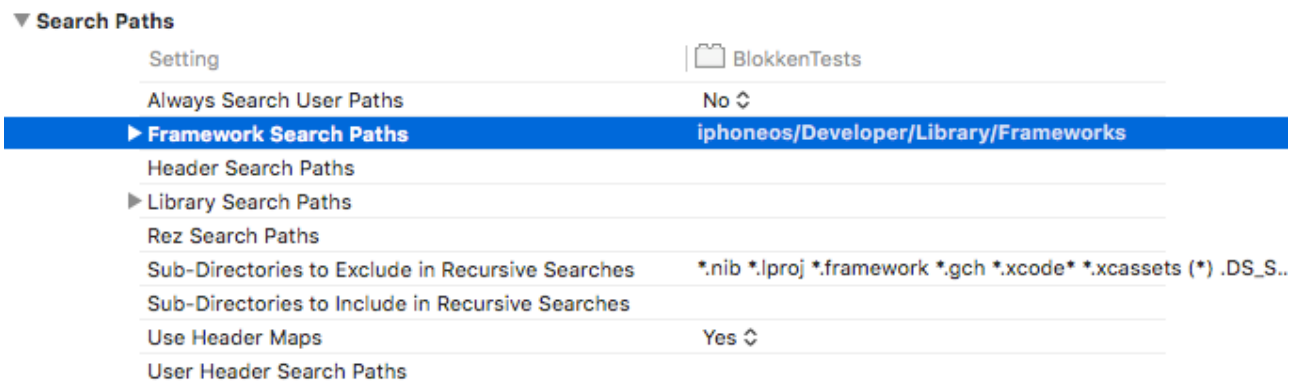
Setting	BlokkenTests
Asset Pack Manifest URL Prefix	
Embed Asset Packs In Product Bundle	No ↕
Enable On Demand Resources	No ↕
On Demand Resources Initial Install Tags	
On Demand Resources Prefetch Order	

Scroll naar beneden, tot je bij de rubriek *Search Paths* bent.

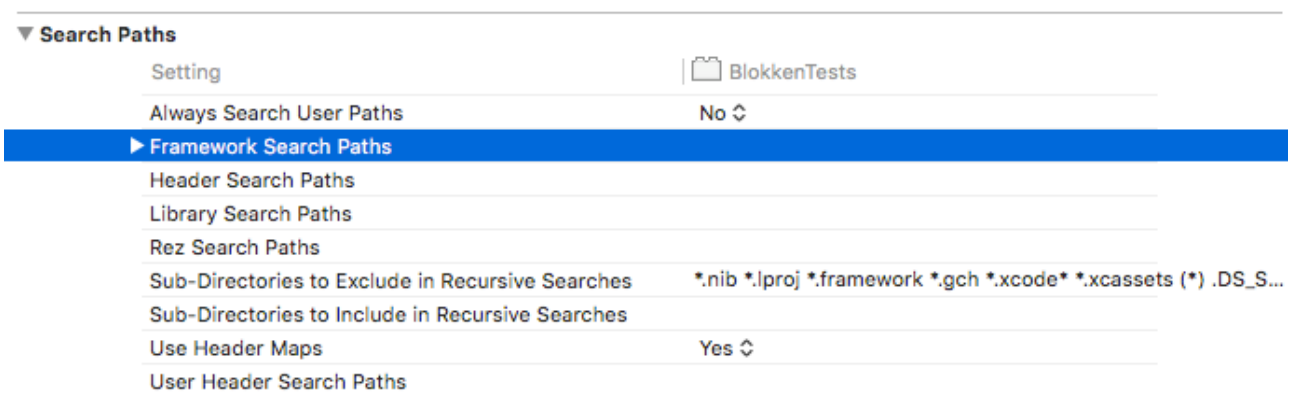
▼ Search Paths

Setting	BlokkenTests
Always Search User Paths	No ↕
Framework Search Paths	iphoneos/Developer/Library/Frameworks
Header Search Paths	
Library Search Paths	
Rez Search Paths	
Sub-Directories to Exclude in Recursive Searches	*.nib *.lproj *.framework *.gch *.xcode* *.xcassets (*) .DS_S...
Sub-Directories to Include in Recursive Searches	
Use Header Maps	Yes ↕
User Header Search Paths	

Klik op de optie *Framework Search Paths* om deze te selecteren:



Druk op de Delete-toets om deze instelling te wissen:



Klaar! Als je de app nu opnieuw start, is de waarschuwing verdwenen.

2 Wijzigingen in Xcode

In Xcode is veel veranderd, maar gelukkig heeft niet alles evenveel impact. De twee belangrijkste wijzigingen zijn:

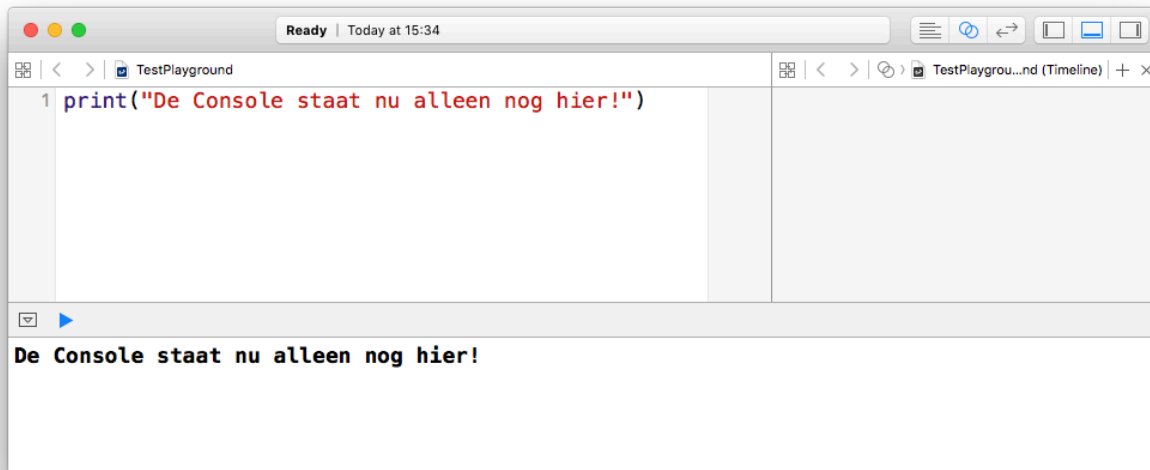
- Playgrounds werken, met name wat het tonen van resultaten (en views) betreft, anders;
- Auto Layout werkt niet langer via menu-opties;

Playgrounds

De Console-uitvoer van Playgrounds is verhuisd

In oudere versies van Swift gebruikte je de Timeline om te kijken wat er, door functies zoals `println()`, naar de Console werd gestuurd. Als je vaak met Playgrounds hebt gewerkt, ken je ongetwijfeld de toetscombinatie `⌘Enter` om snel de Timeline te laten zien...

Die toetscombinatie heb je bij Xcode 7 en Swift 2 een stuk minder vaak nodig. Console-uitvoer wordt nu niet langer in de Timeline getoond, maar alleen nog in het Debug-paneel. Dat paneel krijg je te zien door *View* → *Debug Area* → *Show Debug Area* te kiezen, of door de wat vreemde toetscombinatie `⌘Y` te gebruiken.

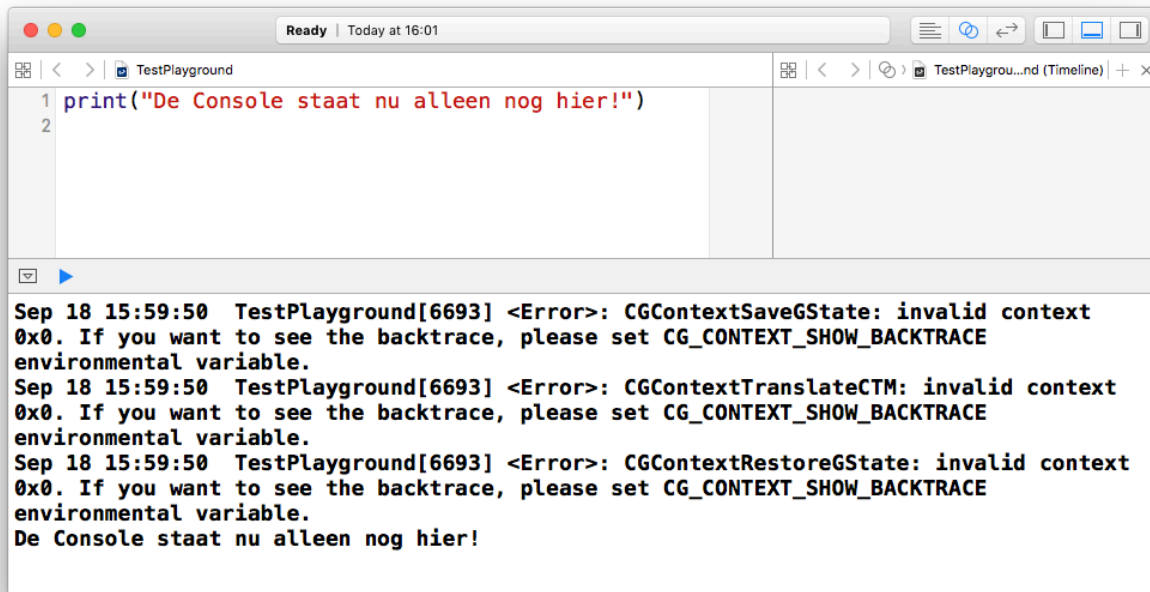


De letter Y wordt in het Engels uitgesproken als "why". Daar komt ook de toetscombinatie voor het Debug-paneel vandaan: je hebt dat paneel immers nodig als je je afvraagt waarom je programmacode niet goed werkt ("WHY ins't my code working?")

Overigens: `println()` bestaat ook niet meer, zoals je in het volgende hoofdstuk zult zien. Om tekst naar de Console te sturen, gebruik je vanaf nu `print()`.

Een vervelende bug: CGContextSaveGState: invalid context

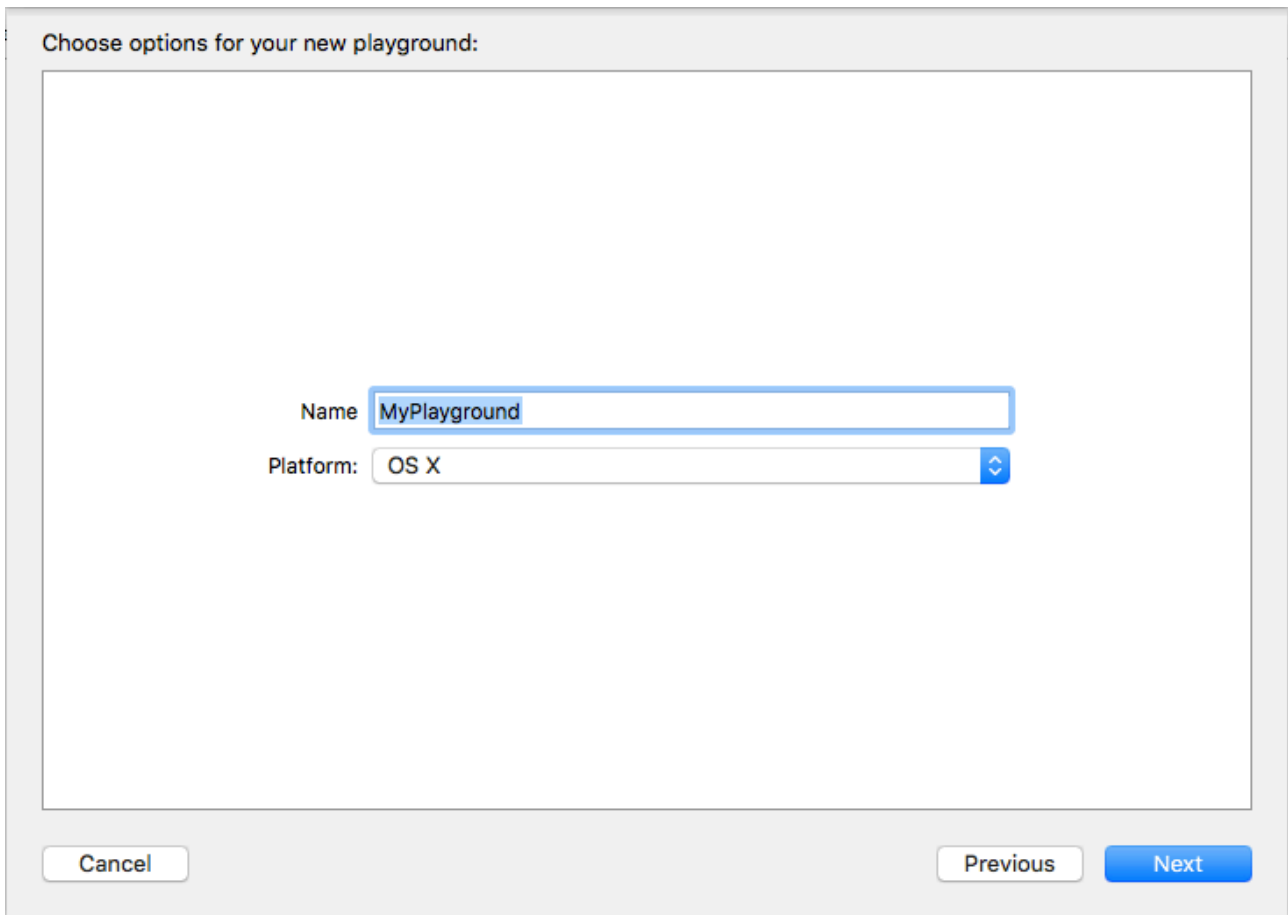
In de eerste versies van Xcode 7 zit een storende fout die, zodra je de Console in het Debug-paneel wilt bekijken, voor een flinke hoop meldingen zorgt, allemaal onder de noemer *CGContextSaveGState: invalid context*.



Eén manier om daar om heen te werken: gebruik geen iOS-Playgrounds maar OS X-Playgrounds. Dat kan op twee manieren.

Oplossing 1: voor nieuwe Playgrounds

Wanneer je een nieuwe Playground maakt, verschijnt het venster uit de volgende afbeelding.

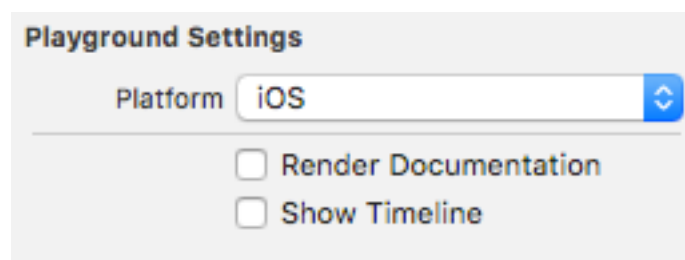


Zorg dat bij *Platform* de optie *OS X* is geselecteerd (en dus niet *iOS*). Je ziet de meldingen dan niet langer in het Debug-paneel verschijnen.

Oplossing 2: voor bestaande Playgrounds

Wanneer je bestaande Playground wordt overstroomd door deze meldingen, kies je *View* → *Utilities* → *Show Utilities*, waarna het Utilities-paneel verschijnt. Ga naar de eerste Tab van dat paneel: de *File Inspector*. Je kunt ook op de toetscombinatie `⌘⌘1` drukken.

In het Utilities-paneel zie je, onder het kopje *Playground Settings*, de mogelijkheid staan om een ander platform te kiezen:



Selecteer OS X. Sluit nu je Playground en open hem opnieuw: de meldingen zijn nu verdwenen.

Views in Playgrounds tonen

In vorige versies van Xcode gebruikte je, uit de module `XCPlayground`, de functie `XCShowView()` om een view (en zijn subviews) op de Timeline te tonen.

De `XCPlayground`-module is sinds Xcode 7.1 flink uitgebreid, maar dat is ten koste gegaan van de `XCShowView()`-functie; die is verouderd en zal op termijn verdwijnen. Bovendien heeft het tweede argument nu een externe naam, dus:

```
XCShowView("Een view", view: hoofdView)
```

in plaats van:

```
XCShowView("Een view", hoofdView)
```

De opvolger van `XCShowView()`: een property

Omdat `XCShowView()` op termijn zal verdwijnen, kun je beter de nieuwe manier gebruiken: de globale property `XCPlaygroundPage.currentPage.liveView`. Als je bijvoorbeeld `hoofdView` op de Timeline wilt tonen, doe je dat als volgt:

```
XCPlaygroundPage.currentPage.liveView = hoofdView
```

Auto Layout en de menu-opties

In de vorige versie van Xcode kon je, om elementen met Auto Layout uit te lijnen, gebruik maken van een paar opties uit het Editor-menu: *Editor → Pin* en *Editor → Align*.

De eerste optie, *Editor → Pin*, is verdwenen. Om elementen uit te lijnen, gebruik je nu het *Pin*-icoon onderaan in het Storyboard:



De tweede optie, *Editor → Align*, doet niets meer met Auto Layout! Hij verschuift je elementen weliswaar op het Storyboard, maar Xcode maakt niet langer Auto Layout-constraints voor die elementen aangemaakt.

Inderdaad: de opties *Editor → Align → Horizontally in Container* en *Editor → Align → Vertically in Container* zijn, als je Auto Layout constraints wilt toevoegen, dus zinloos geworden. In plaats daarvan gebruik je nu het *Align*-icoon, onderaan in het Storyboard:



Je kunt nog wel ^-slepen (of slepen terwijl je de rechter muisknop ingedrukt houdt) om Auto Layout-constraints toe te voegen.

3 Wijzigingen in Swift

Het aantal wijzigingen in Swift 2 is enorm groot. Als je al apps of Playgrounds in Swift 1 hebt geschreven, krijg je er dan ook meteen mee te maken: je programmacode werkt helemaal niet meer en je ziet alleen nog maar foutmeldingen en waarschuwingen.

De meeste problemen kun je gelukkig oplossen als je rekening houdt met de volgende wijzigingen, die we in dit hoofdstuk uitgebreid bespreken.

- `println()` is vervangen door `print()`;
- Xcode houdt mogelijke constanten goed in de gaten;
- `do while` is vervangen door `repeat while`;
- `countElements()` bestaat niet meer;
- Strings werken anders;
- Getallen omzetten naar andere datatypes: veel wijzigingen;
- Functies: externe argumentnamen werken anders

`println()` is vervangen door `print()`

In versie 1 van Swift bestonden er twee functies om tekst naar de Console te sturen: `println()` en `print()`. De `println()`-functie zette een 'harde return' achter je tekst, zodat de volgende tekst op een nieuwe regel begon. Als je geen harde return wilde, gebruikte je `print()`.

In Swift 2 bestaat `println()` niet meer. Je gebruikt nu dus altijd `print()`. Na `print()` wordt in principe altijd op een nieuwe regel begonnen.

In dit voorbeeld zie je hoe dit werkt:

```
print("Regel 1")
print("Regel 2")
```



Zoals je ziet, wordt in Xcode 7 geen Timeline meer gebruikt om de inhoud van de Console te tonen: dat gebeurt nu in het Debug-paneel. Kies View → Debug Area → Show Debug Area of druk op `⇧⌘Y` om het Debug-paneel te openen.

Als je niet wilt dat er op een nieuwe regel wordt, begonnen, gebruik je een extra parameter, met de naam *terminator*. Daarachter kun je opgeven welk teken er in plaats van een 'harde return' moet worden gebruikt om tekst van elkaar te scheiden. In het volgende voorbeeld wordt daarvoor een spatie gebruikt:

```
print("Regel 1", terminator: " ")
print("en nog steeds Regel 1")
print("Regel 2")
```

```
1 // Tekst na elkaar:
2 print("Regel 1", terminator: " ")
3 print("en nog steeds Regel 1")
4 print("Regel 2")
5
```

Regel 1
en nog steeds Regel 1
Regel 2

Je kunt ook een lege *terminator*-string opgeven:

```
print("a", terminator: "")
print("b") // Wat hierna komt, begint op een nieuwe regel
print("c")
```

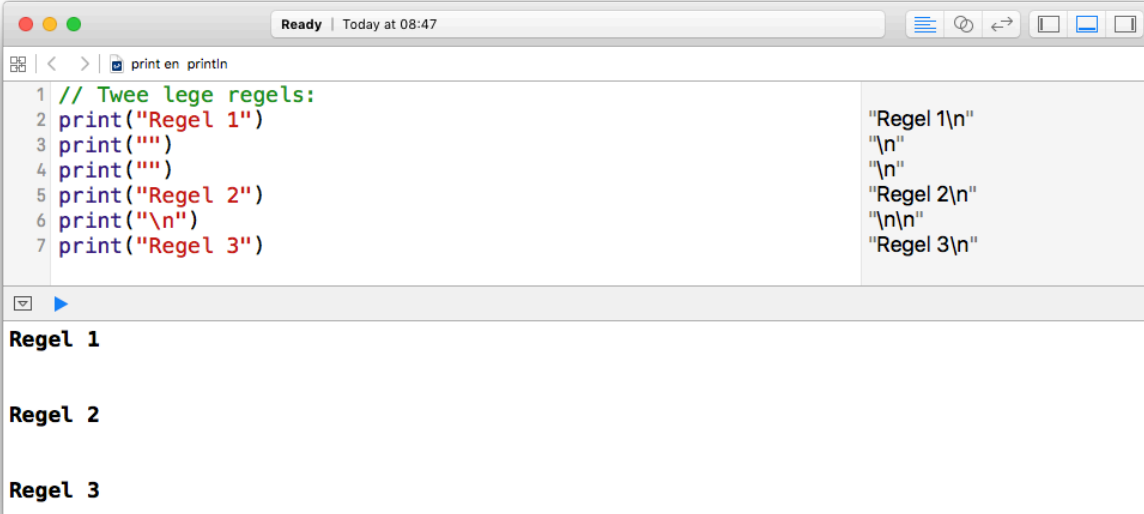
```
1 // Tekst na elkaar, zonder spatie:
2 print("a", terminator: "")
3 print("b") // Wat hierna komt, begint op een nieuwe regel
4 print("c")
5
```

a
b
c

Wil je twee lege regels? Dat kan met twee afzonderlijke `print()`-functies. Je kunt ook gebruik maken van de speciale 'nieuwe regel'-code: `\n`.

```
print("Regel 1")
print("")
print("")
print("Regel 2")
```

```
print("\n")
print("Regel 3")
```



The screenshot shows a code editor window titled "Ready | Today at 08:47". The editor contains a Python script with the following code:

```
1 // Twee lege regels:
2 print("Regel 1")
3 print("")
4 print("")
5 print("Regel 2")
6 print("\n")
7 print("Regel 3")
```

To the right of the code, the output is displayed as a string representation: "Regel 1\n", "\n", "\n", "Regel 2\n", "\n\n", and "Regel 3\n". Below the code editor, there is a console window showing the output of the script:

```
Regel 1

Regel 2

Regel 3
```

Nieuw bij print(): meer argumenten

De `print()`-functie kan nu meer argumenten tegelijk aan. Deze argumenten worden na elkaar afgedrukt, met telkens een spatie ertussen:

```
print("Argument 1", "Argument 2", "Argument 3")

let a = 1, b = 2, c = 3
print(a, b, c)
```

```
1 // Meer dan één argument:
2 print("Argument 1", "Argument 2", "Argument 3")
3
4 // Meer dan één variabele:
5 let a = 1, b = 2, c = 3
6 print(a, b, c)
7 // De 'oude' manier, met interpolatie:
8 print("\a) \b) \c)")
9
```

Argument 1 Argument 2 Argument 3
1 2 3
1 2 3

Je kunt ook opgeven welk scheidingsteken er tussen de diverse argumenten moet worden geplaatst. Zo kun je bijvoorbeeld komma's tussen de diverse argumenten laten afdrukken, of je kunt elk argument op een nieuwe regel laten beginnen:

```
let a = 1, b = 2, c = 3

// Een komma en spatie tussen elk argument:
print(a, b, c, separator:", ")

// Na elk argument op een nieuwe regel beginnen:
print(a, b, c, separator:"\n")
```

```
1 let a = 1, b = 2, c = 3
2
3 // Een komma en spatie tussen elk argument,
4 // aan het einde een harde return:
5 print(a, b, c, separator:", ")
6
7 // Na elk argument een harde return:
8 print(a, b, c, separator:"\n")
9
```

1, 2, 3
1
2
3

print() en lege regels

Met de combinatie `\n` kun je opgeven dat er op een nieuwe regel moet worden begonnen. Om alleen een lege regel af te drukken, kun je ook een 'lege' string aan `print()` doorgeven:

```
// Een lege regel:  
print("")  
  
// Drie lege regels:  
// (twee voor elke \n en  
// één omdat dat altijd gebeurt na print()  
print("\n\n")
```

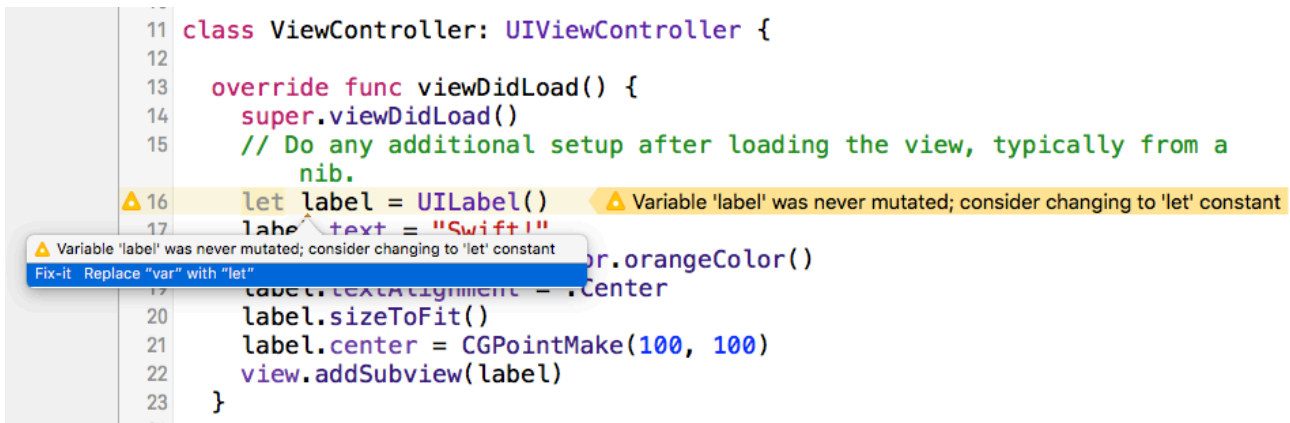
Xcode houdt mogelijke constanten goed in de gaten

Eén van de kenmerken van Swift is veiligheid. Een voorbeeld daarvan zijn de constanten: variabelen waarvan de inhoud, nadat ze zijn geïnitieerd, niet meer kan worden gewijzigd. Probeer je dat toch dan krijg je een foutmelding.

Wanneer je de inhoud van een variabele nooit meer wijzigt nadat je de variabele hebt gemaakt, verdient het dan ook de voorkeur om er een constante van te maken (met het trefwoord `let`):

```
var a = 10 // een variabele die kan worden gewijzigd  
let c = 11 // een constante: kan niet meer worden gewijzigd
```

Sinds versie 2 van Swift helpt Xcode je daarbij: als het merkt dat een variabele eigenlijk net zo goed een constante zou kunnen zijn, krijg je een waarschuwing te zien met het advies om `let` te gebruiken in plaats van `var`.



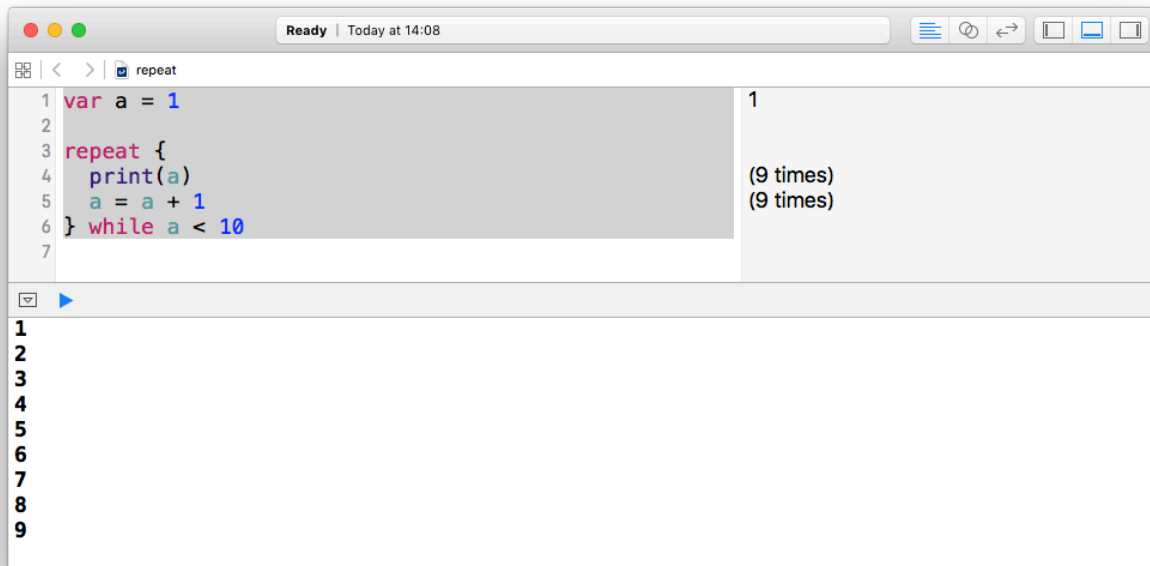
do while is vervangen door repeat while

De eerste versies van Swift beschikten over vier methoden om programmacode herhaaldelijk uit te voeren: `while()`, `do while()`, `for in` en `for`. Die methoden zijn in principe niet gewijzigd in Swift 2, op een kleinigheid na: het trefwoord `do` is veranderd in `repeat`. Dit is gedaan om in één oogopslag duidelijk te maken wat er precies gebeurt: "herhaal de volgende opdrachten zolang deze voorwaarde 'waar' is." Een voorbeeld zie je in de volgende Playground:

"herhaal de regels waarin de inhoud van `a` in de Console wordt afgedrukt en `a` met één wordt verhoogd, zolang `a` kleiner is dan 10".

```
var a = 1

repeat {
    print(a)
    a = a + 1
} while a < 10
```

countElements() bestaat niet meer

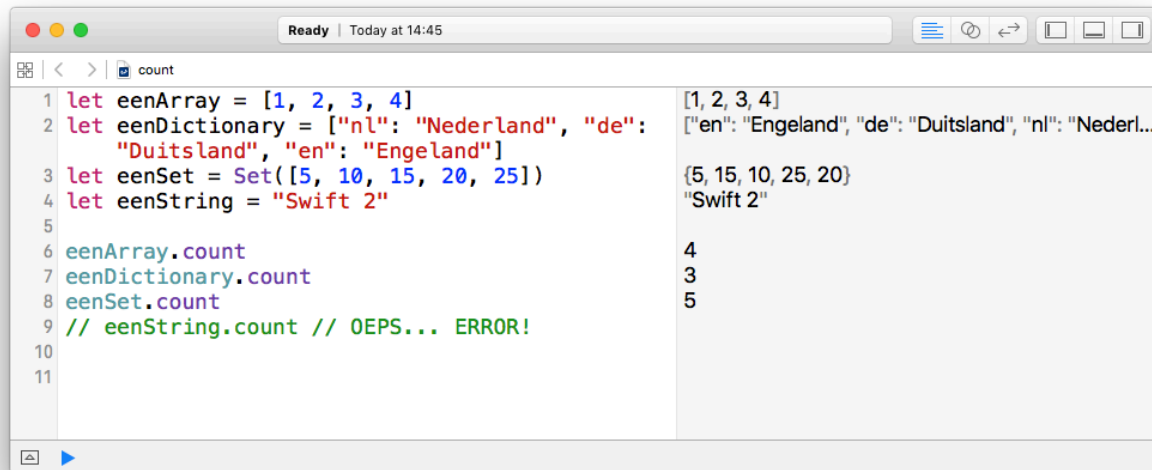
In Swift 1 kon je, met behulp van de globale functie `countElements()`, opvragen uit hoeveel elementen een String, Dictionary, Array, Set of andere collectie bestond.

Allereerst: Strings zijn niet langer meer collecties - daar komen we dadelijk nog op terug. Daar komt nog bij dat de functie `countElements()` niet langer meer bestaat.

In plaats daarvan beschikt elke collectie nu over een `.count`-property (let op: een property, géén methode). In de volgende Playground zie je een paar voorbeelden.

```
let eenArray = [1, 2, 3, 4]
let eenDictionary = ["nl": "Nederland", "de": "Duitsland", "en": "Engeland"]
let eenSet = Set([5, 10, 15, 20, 25])
let eenString = "Swift 2"

eenArray.count
eenDictionary.count
eenSet.count
eenString.count // OEPS... ERROR!
```



Zoals je ziet, werkt de `.count`-property prima - behalve bij `String`-waarden. Strings zijn namelijk niet langer collecties; Swift 2 ziet ze als één entiteit en niet als verzameling van `Character`-waarden.

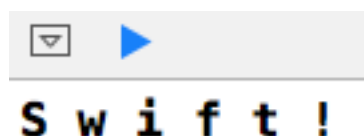
Strings werken anders

Strings zijn geen collecties meer. Je kunt dus niet met `for in` door een string heenlopen. In plaats daarvan gebruik je een nieuwe property waarover elke `String` beschikt: `.characters`. Deze property retourneert een collectie met alle elementen uit de string, waarbij elk element het datatype `character` heeft. Een voorbeeld:

```
let str = "Swift!"
let collectieMetCharacters = str.characters

for eenCharacter in collectieMetCharacters {
    print(eenCharacter, terminator: " ")
}
```

In deze Playground wordt door van `.characters` afkomstige collectie heengelopen: elk teken wordt, gescheiden door een spatie, in de Console weergegeven.



Omdat strings geen collecties meer zijn, hebben ze ook geen `.count`-property. Ook hier is de oplossing: gebruik de `.characters`-property om een reeks met Character-waarden op te halen en gebruik voor die collectie de `.count`-property:

```
let str = "Swift!"

let collectieMetCharacters = str.characters

// str.count // FOUT!

// In plaats daarvan:
collectieMetCharacters.count

// of:
str.characters.count

// of:
"Swift!".characters.count
```



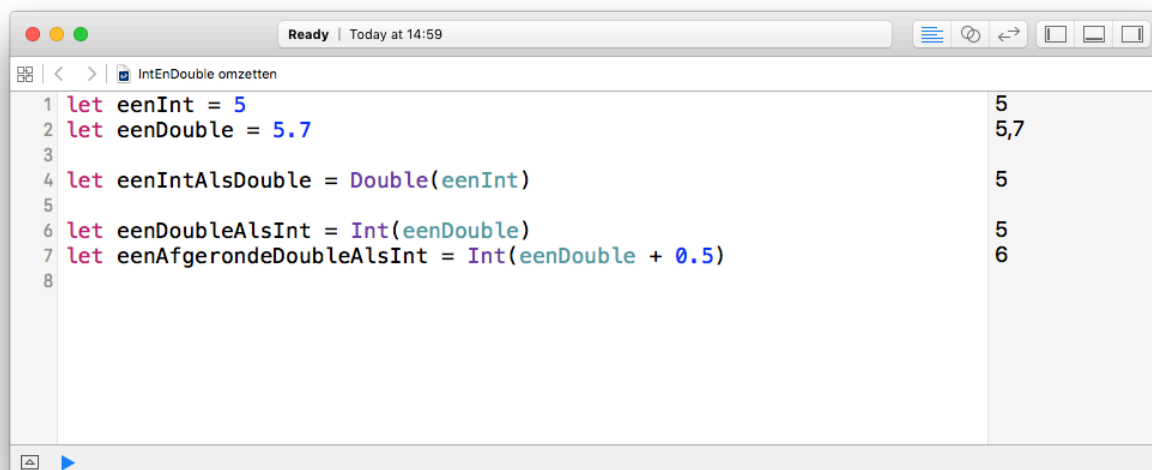
Getallen omzetten naar andere datatypes: veel wijzigingen

Om het datatype van een getal om te zetten naar een ander datatype, kun je in Swift 2 gebruik maken van nieuwe initializers. Zo kun je met de initializer `Int()` een `Float`-waarde omzetten in `Int`; omgekeerd kun je met `Float()` een `Int`-waarde omzetten in een waarde met het datatype `Float`. De methode `.toInt()` is daarmee overbodig geworden en is dan ook verwijderd.

```
let eenInt = 5
let eenDouble = 5.7

let eenIntAlsDouble = Double(eenInt)

let eenDoubleAlsInt = Int(eenDouble)
let eenAfgerondeDoubleAlsInt = Int(eenDouble + 0.5)
```



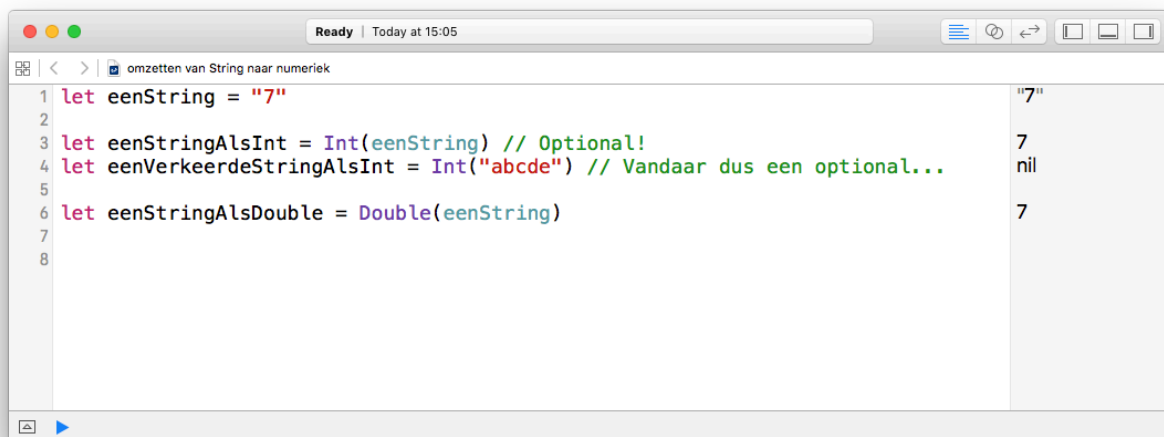
Je ziet dat Double-waarden niet worden afgerond als je ze omzet naar een Int-waarde. Vandaar een klassieke truc (die ik zelf ooit heb geleerd van Luc Volders - degene die mij ooit m'n allereerste computer verkocht, een VIC-20): om een getal af te ronden op een geheel getal, tel je er 0,5 bij op en verwijder je vervolgens alles achter de komma. Luc, als je dit leest: het is en blijft een gouden tip!

Nieuw in Swift is dat je nu ook een String-waarde op een gemakkelijke manier kunt omzetten in een Int, Float, Double enzovoort. Er is één kleinigheidje om rekening mee te houden: je krijgt een optional terug. De volgende Playground demonstreert dit:

```
let eenString = "7"

let eenStringAlsInt = Int(eenString) // Optional!
let eenVerkeerdeStringAlsInt = Int("abcde") // Vandaar dus een optional...

let eenStringAlsDouble = Double(eenString)
```



Functies: externe argumentnamen werken anders

In Swift 1 was het werken met externe argumentnamen niet altijd gemakkelijk: wellicht herinner je je die hekjes nog die je soms wel en soms niet moest gebruiken... Een bron van verwarring!

In Swift 2 is dat allemaal opgelost. Wanneer je met functie-argumenten werkt, zijn er nog maar een paar eenvoudige regels:

- Het eerste argument is nu ALTIJD naamloos. Wil je tóch een externe argumentnaam, dan zet je die, bij de functiedeclaratie, vóór de interne-argumentnaam.

- Elk volgende argument heeft een externe argumentnaam. In principe is die gelijk aan de interne argumentnaam. Als je een andere externe argumentnaam wilt gebruiken, zet je die vóór de interne argumentnaam. Wil je voor een tweede of volgende element géén externe argumentnaam gebruiken? Zet dan een _ vóór de interne argumentnaam.

De volgende Playground laat zien hoe je, bij het definiëren van je functies, precies kunt bepalen of een argument wél of geen externe naam heeft (en of dat argument, bij het aanroepen van de functie, dus wel of niet moet worden 'genoemd').

```
//: # (Externe) argumentnamen
//: ### De 'standaard'-definities
/*:
Eén argument? Dan hoef je bij het aanroepen geen naam op te geven
*/
func eenArgument(a: Int) {}
eenArgument(1)

/*:
Twee argumenten?
Voor het eerste argument geef je bij het aanroepen
geen naam op.
Voor het tweede argument wél.
*/
func tweeArgumenten(a: Int, b: Int) {}
tweeArgumenten(1, b: 2)

/*:
Drie argumenten?
Voor het eerste argument geef je bij het aanroepen
geen naam op.
Voor het tweede en elk volgende argument wél.
*/
func drieArgumenten(a: Int, b: Int, c: Int) {}
drieArgumenten(1, b: 2, c: 3)

//: ### Aparte definities
/*:
Eén argument met een externe naam?
Zet, bij de functiedefinitie, die externe naam vóór de 'interne'
argumentnaam. Dat mag dezelfde naam zijn als de 'interne' naam,
maar dat hoeft niet.
Bij het aanroepen van de functie moet je nu dus,
zelfs bij een eerste argument, die externe naam opgeven.
*/
func eenArgumentMetExterneNaam(a a: Int) {}
eenArgumentMetExterneNaam(a: 1)

func eenArgumentMetAndereExterneNaam(aAnders a: Int) {}
eenArgumentMetAndereExterneNaam(aAnders: 1)

/*:
Een functiedefinitie met twee argumenten, allebei zonder naam?
```

```

Bij het eerste argument hoef je niets te doen.
Bij het tweede argument: zet er een underscore vóór.
*/
func tweeArgumentenZonderNamen(a: Int, _ b: Int) {}
tweeArgumentenZonderNamen(1, 2)

/*:
Een functiedefinitie met twee argumenten,
allebei met een externe naam?
Bij het eerste argument: zet de externe naam vóór het argument.
Bij het tweede argument hoef je niets te doen.
*/
func tweeArgumentenMetNamen(a a: Int, b: Int) {}
tweeArgumentenMetNamen(a: 1, b: 2)

/*:
Wil je een argument een andere externe naam geven?
Zet die naam dan vóór het argument.
*/
func tweeArgumentenMetAndereNamen(aAnders a: Int, bAnders b: Int) {}
tweeArgumentenMetAndereNamen(aAnders: 1, bAnders: 2)

```

4 Je volgende stap

Kom je er niet helemaal uit? Aarzel dan niet en bezoek het forum van de iOS Academie: www.iosacademie.nl/forum. We helpen je daar graag verder!

Nog een tip: wellicht ken je de eBook-serie Apps bouwen met Swift al. Je leert, ook als je nog geen programmeerervaring hebt, snel en grondig hoe je zelf apps kunt maken voor je iPhone en iPad. In deze serie verschijnen regelmatig nieuwe delen. Bovendien... als je al één of meer van die eBooks hebt aangeschaft: alle updates zijn levenslang gratis! Vergeet dus niet om je gratis updates te downloaden!

Meer informatie over deze populaire eBooks vind je hier:

www.iosacademie.nl/apps-bouwen-met-swift

